Unit 1 Project - Frictionless Reproducibility (due by 11pm on 9-23)

Goal. Transform your existing data analysis from a disorganized state into a well-structured, well-documented, reproducible, and maintainable project.

Learning Objectives. By completing this project, you will:

- Organize messy code into structured, maintainable projects.
- Automate your workflow to make reproducibility effortless.

Project Requirements

Core Requirements

1. Source Material Selection Choose your own previous data analysis project (Jupyter notebook, python script, existing repo, etc.). It can be from a course or from your research. Before you do anything to your existing project, either start a GitHub repository (if you already know how) or zip its contents, so we can see how the project started out.

2. Project Structure Implement a clean, logical directory structure such as:

```
project-name/
├── data/
│   ├── raw/          # immutable source (read-only)
│   └── processed/    # cleaned data
├── src/
│   ├── pipeline/     # populates processed/ and artifacts/
│   └── analysis/     # populates results/
├── artifacts/        # intermediate outputs
│   ├── sufficient-stats/
│   └── models/
├── results/
│   ├── tables/
│   ├── figures/
│   └── report/
├── tests/
├── docs/
├── requirements.txt (or environment.yml)
├── run_analysis.py (or Makefile)
├── README.md
└── .gitignore
```

3. Version Control

- Initialize git repository with meaningful initial commit
- Create at least 5 commits showing progressive improvement
- Write clear, descriptive commit messages following conventional format
- Include some relevant files in your .gitignore

4. Environment Management

- Create isolated virtual environment
- Generate comprehensive requirements/dependencies file
- Include clear setup instructions that work on a fresh system

5. Documentation Create a professional README.md including:

- Clear project description and objectives
- Installation/setup instructions
- Usage examples with expected outputs

6. Testing Strategy Implement at least 2 meaningful tests:

- Data validation (e.g., expected columns, value ranges, no missing critical data)
- Function correctness (e.g., calculation accuracy, expected output format)
- Pipeline integrity (e.g., end-to-end run produces expected files)
- Statistical validation (e.g., assessing approximate type-I error control)

7. One-Command Execution Create a single entry point to run the complete analysis:

- `python run_analysis.py` or `make all` or similar
- Should handle the full pipeline from raw data to final outputs
- Include progress indicators or logging
- Generate all figures, tables, and summary statistics

Submission Requirements

1. **Repository URL** with public access
2. **Reflection Document** (at most one page) addressing:
   - What was the original state of your analysis?
   - What were the biggest challenges in the transformation?
   - Which improvements had the most impact on reproducibility?
   - What would you do differently in a future project?
   - How long did each major component take to implement?

Suggestions

Timeline

Week 1: Select source material, plan structure.
Week 2: Create directory structure.
Week 3: Start version control and isolating dependencies.
Week 4: Refactor code, add documentation, create tests. Write reflection.

Tools

- **Version Control**: `git` with GitHub
- **Environment**: `venv`
- **Documentation**: Markdown rendered on GitHub, plus proper pep257 docs in your code.
- **Testing**: `pytest`

Tips

- Start with messy code
- Commit as you go - separate commits for separate tasks
  - Avoid generic commit messages like "update file"
- Test your setup instructions on a different computer/environment
- Ask for feedback during development
- Keep notes on challenges for your reflection document
- Test the one-command execution
- Remember that documentation is written for someone else, not just you
- You do not have to use a project written in `python`
- You don't have to include the data if it's too large. If you don't you should definitely save the requisite outputs of your analysis (in `artifacts/`) needed to reproduce the results. In that case, you should have two separate scripts: one end-to-end script that I could run with access to the full data, and one that I can run to reproduce the results assuming I don't have the full data.